Custom plotting and related functions for compositional data analysis

Michael F. Charlton[1]

30 July 2017

[1]UCL Qatar

pkde2d --- point density estimation

The **pkde2d()** function is a minor adaptation of the **kde2d()** function found in the MASS package (Venables and Ripley, 2002). Its purpose is to estimate the density of a single datum following 2-dimensional kernel density estimation. It permits the construction of density contours containing a specified percentage of the specimens in a given sample.

Function arguments:

- *x = x coordinate of data*
- *y = y coordinate of data*
- *h = vector of bandwiths for x and y*
- *n = number of grid points*
- *lims = limits of the rectangle covered by the grid*

Function values:

- *x = the x coordinates of the input data*
- *y = the y coordinates of the input data*
- *z = n x n matrix of estimated density*

*See **kde2d()** in package MASS (Venables and Ripley, 2002)*

```
pkde2d<-function (x, y, h, n = 25, lims = c(range(x), range(y)))
{
  nx <- length(x)
  if (length(y) != nx)
    stop("data vectors must be the same length")
  if (any(!is.finite(x)) || any(!is.finite(y)))
    stop("missing or infinite values in the data are not allowed")
```

```
  if (any(!is.finite(lims)))
    stop("only finite values are allowed in 'lims'")
  n <- rep(n, length.out = 2L)
  h <- if (missing(h))
    c(bandwidth.nrd(x), bandwidth.nrd(y))
  else rep(h, length.out = 2L)
  h <- h/4
  ax <- outer(x, x, "-")/h[1L]
  ay <- outer(y, y, "-")/h[2L]
  z <- tcrossprod(matrix(dnorm(ax), , nx), matrix(dnorm(ay),
                                                  , nx))/(nx * h[1L] * h[2L])
  list(x = x, y = y, z = z)
}
```

geomean()---calculation of th geometric mean for a given specimen

The **geomean()** function calculates the geometric mean for a specimen or set of specimens. It is essential for performing centred-log ratio transformations of a dataset.

Function arguments:

- *x = a vector of numeric values*

```
geomean<-function(x){
  log_x<-log(x)
  gm<-exp(mean(log_x))
  return(gm)
}
```

pcaplot()---standard PCA plot

This function generates a PCA plot of standardized data with loadings and scores normalized to the eigenvalues.

Function arguments:

- *x = a matrix or dataframe of numeric values*
- *xx = PC axis to plot on the abscissa (1, 2, 3, ...)*
- *yy = PC axis to plot on the ordinate (1, 2, 3, ...)*
- *group = a vector for defining associated specimens (members of the same group will be assigned the same colour)*

- *label = a vector of names to plot to the upper right of each point*

Function Values:

- *clusters = data clustered by grouping variable*
- *pc = results of the PCA by* ***prcomp()***
- *scores = a dataframe of specimen scores arising form the PCA*
- *norm.scores = a dataframe of eigenvalue normalized scores*
- *fc = a vector of field colours assigned to each group*

```
pcaplot<-function(x,xx,yy,group,label)
{
  pca<-prcomp(x,retx=T,scale=T)

  scores<-array(data=NA,dim=c(nrow(pca$x),ncol(pca$x)),
                dimnames=list(rownames(pca$x),colnames(pca$x)))
  for (l in 1:ncol(pca$x)){
    scores[,l]<-pca$x[,l]/pca$sdev[l]
  }


  loadings<-array(data=NA,dim=c(nrow(pca$rotation),
                          ncol(pca$rotation)),
                  dimnames=list(rownames(pca$rotation),
                                colnames(pca$rotation)))

  for (m in 1:ncol(pca$rotation)){
    loadings[,m]<-pca$rotation[,m]*pca$sdev[m]
  }
  percx<-round(pca$sdev[xx]^2/sum(pca$sdev^2)*100,2)
  percy<-round(pca$sdev[yy]^2/sum(pca$sdev^2)*100,2)

  clusters<-list()
  for(i in 1:nlevels(group)){
    clusters[[i]]<-scores[group==levels(group)[i],]
  }

  fc<-rainbow(length(clusters),s=1,v=1,alpha=0.3)
  plot(scores[,xx],scores[,yy],
       pch=NA,
       xlim=c(min(loadings[,xx],scores[,xx])-0.5,
              max(loadings[,xx],scores[,xx])+0.5),
       ylim=c(min(loadings[,yy],scores[,yy])-0.5,
```

```
              max(loadings[,yy],scores[,yy])+0.5),
       xlab=paste(colnames(scores)[xx]," (",percx,"%)",sep=""),
       ylab=paste(colnames(scores)[yy]," (",percy,"%)",sep="")
  )

  for(g in 1:length(clusters)){
    if(is.null(nrow(clusters[[g]]))==F){
      points(clusters[[g]][,xx],clusters[[g]][,yy],
             pch=21,
             cex=2,
             col=NA,
             bg=fc[g])}
    else{
      points(clusters[[g]][xx],clusters[[g]][yy],
             pch=21,
             cex=2,
             col=NA,
             bg=fc[g])
    }
  }
  arrows(0,0,loadings[,xx],loadings[,yy],
         length=0.1,
         angle=15
  )
  text(loadings[,xx]*1.1,loadings[,yy]*1.1,rownames(loadings),
       cex=0.8,
       col="darkblue",
       font=2
  )
  text(scores[,xx]+0.1,scores[,yy]+0.1,label,cex=0.6,adj=0.25)

  list(clusters=clusters, pc=pca, scores=pca$x, norm.scores=scores, fc=fc)

}
```

pcaplot.clr() --- unstandardardized PCA plot

This function generates a PCA plot of unstandardized data with loadings and scores normalized to the eigenvalues. It is designed for handling clr-transformed data, though loading vectors may require alternative scaling.

Function arguments:

- *x = data*
- *xx = PC axis to plot on the abscissa (1, 2, 3, ...)*
- *yy = PC axis to plot on the ordinate (1, 2, 3, ...)*
- *group = a vector for defining associated specimens (members of the same group will be assigned the same colour)*
- *label = a vector of names to plot to the upper right of each point*

Function Values:

- *clusters = data clustered by grouping variable*
- *pc = results of the PCA by* ***prcomp()***
- *scores = a dataframe of specimen scores arising form the PCA*
- *norm.scores = a dataframe of eigenvalue normalized scores*
- *fc = a vector of field colours assigned to each group*

```
pcaplot.clr<-function(x,xx,yy,group,label)
{
  pca<-prcomp(x,retx=T,scale=F)

  scores<-array(data=NA,dim=c(nrow(pca$x),ncol(pca$x)),
                dimnames=list(rownames(pca$x),colnames(pca$x)))
  for (l in 1:ncol(pca$x)){
    scores[,l]<-pca$x[,l]/pca$sdev[l]
  }


  loadings<-array(data=NA,dim=c(nrow(pca$rotation),
                                ncol(pca$rotation)),
                  dimnames=list(rownames(pca$rotation),
                                colnames(pca$rotation)))

  for (m in 1:ncol(pca$rotation)){
    loadings[,m]<-pca$rotation[,m]*pca$sdev[m]
  }
  percx<-round(pca$sdev[xx]^2/sum(pca$sdev^2)*100,2)
  percy<-round(pca$sdev[yy]^2/sum(pca$sdev^2)*100,2)

  clusters<-list()
  for(i in 1:nlevels(group)){
    clusters[[i]]<-scores[group==levels(group)[i],]
  }
```

```
  fc<-rainbow(length(clusters),s=1,v=1,alpha=0.3)
  plot(scores[,xx],scores[,yy],
       pch=NA,
       xlim=c(min(loadings[,xx],scores[,xx])-0.5,
              max(loadings[,xx],scores[,xx])+0.5),
       ylim=c(min(loadings[,yy],scores[,yy])-0.5,
              max(loadings[,yy],scores[,yy])+0.5),
       xlab=paste(colnames(scores)[xx]," (",percx,"%)",sep=""),
       ylab=paste(colnames(scores)[yy]," (",percy,"%)",sep="")
  )

  for(g in 1:length(clusters)){
    if(is.null(nrow(clusters[[g]]))==F){
      points(clusters[[g]][,xx],clusters[[g]][,yy],
             pch=21,
             cex=2,
             col=NA,
             bg=fc[g])}
    else{
      points(clusters[[g]][xx],clusters[[g]][yy],
             pch=21,
             cex=2,
             col=NA,
             bg=fc[g])
    }
  }
  arrows(0,0,loadings[,xx],loadings[,yy],
         length=0.1,
         angle=15
  )
  text(loadings[,xx]*1.1,loadings[,yy]*1.1,rownames(loadings),
       cex=0.8,
       col="darkblue",
       font=2
  )
  text(scores[,xx]+0.1,scores[,yy]+0.1,label,cex=0.6,adj=0.25)

  list(clusters=clusters, pc=pca, scores=pca$x, norm.scores=scores, fc=fc)

}
```

pcaplot.kde()---standard PCA plot with KDEs

This function generates a PCA plot of standardized data with loadings and scores normalized to the eigenvalues. It also generates 100% KDE contours around defined training sets.

Function arguments:

- *x = data*
- *xx = PC axis to plot on the abscissa (1, 2, 3, ...)*
- *yy = PC axis to plot on the ordinate (1, 2, 3, ...)*
- *ts = a vector for defining specimen groups or tranings sets (members of the same group will be assigned the same colour)*
- *fc = a vector of field colour choices points and 2D KDE contours for each training set*

Function Values:

- *groups = data clustered by grouping variable*
- *pc = results of the PCA by* ***prcomp()***

```
pcaplot.kde<-function(x,xx,yy,ts,fc)
{
  pca<-prcomp(x,retx=T,scale=T)
  scores<-array(data=NA,
                dim=c(nrow(pca$x),ncol(pca$x)),
                dimnames=list(rownames(pca$x),colnames(pca$x))
                )
  for (l in 1:ncol(pca$x)){
    scores[,l]<-pca$x[,l]/pca$sdev[l]
  }
  loadings<-array(data=NA,
                  dim=c(nrow(pca$rotation),ncol(pca$rotation)),
                  dimnames=list(rownames(pca$rotation),colnames(pca$rotation)
)
                  )
  for (m in 1:ncol(pca$rotation)){
    loadings[,m]<-pca$rotation[,m]*pca$sdev[m]
  }
  percx<-round(pca$sdev[xx]^2/sum(pca$sdev^2)*100,2)
  percy<-round(pca$sdev[yy]^2/sum(pca$sdev^2)*100,2)

  groups<-list()
  for(i in 1:nlevels(ts)){
    groups[[i]]<-subset(scores,ts==levels(ts)[i])
```

```
  }
  widths<-array(data=NA,c(length(groups),2))
  for(j in 1:length(groups)){
    ifelse(nrow(groups[[j]])<4,
           widths[j,1]<-NA,
           widths[j,1]<-width.SJ(groups[[j]][,xx],method="dpi")
    )
    ifelse(nrow(groups[[j]])<4,
           widths[j,2]<-NA,
           widths[j,2]<-width.SJ(groups[[j]][,yy],method="dpi")
    )
  }
  kde<-list()
  for(k in 1:length(groups)){
    if(is.na(widths[k,1])){
      kde[[k]]<-NA
    }
    else {
      kde[[k]]<-kde2d(as.vector(groups[[k]][,xx]),as.vector(groups[[k]][,yy])
,
                      h=c(widths[k,1],widths[k,2]),
                      n=c(1000,1000),
                      lims=c(range(scores[,xx]-5,scores[,xx]+5),
                             range(scores[,yy]-5,scores[,yy]+5))
      )
    }
  }

  pkde<-list()
  for(p in 1:length(groups)){
    if(is.na(widths[p,1])){
      pkde[[p]]<-NA
    }
    else {
      pkde[[p]]<-pkde2d(groups[[p]][,xx],groups[[p]][,yy],
                        h=c(widths[p,1],widths[p,2]),
                        n=c(1000,1000),
                        lims=c(range(scores[,xx]-5,scores[,xx]+5),
                               range(scores[,yy]-5,scores[,yy]+5))
      )
    }
  }
```

```
qe<-seq(0,1,len=10001)

quant<-list()
n_groups<-NULL
for (p in 1:length(groups)){
  if(is.na(widths[p,1])){
    quant[[p]]<-NA
  }
  else {
    n_groups[p]<-nrow(groups[[p]])
    ptdens<-NULL
    for(k in 1:n_groups[p]){
      ptdens<-rbind(ptdens,pkde[[p]]$z[k,k])
    }
    quant[[p]]<-quantile(ptdens,probs=qe)
  }
}

plot(scores[,xx],scores[,yy],
     pch=NA,
     xlim=c(min(scores[,xx])-0.5,max(scores[,xx])+0.5),
     ylim=c(min(scores[,yy])-0.5,max(scores[,yy])+0.5),
     xlab=paste(colnames(scores)[xx]," (",percx,"%)",sep=""),
     ylab=paste(colnames(scores)[yy]," (",percy,"%)",sep="")
)

contours<-list()
for(g in 1:length(groups)){
  if(is.na(widths[g,1])){
    next
  }
  else {
    contours[[g]]<-contourLines(kde[[g]]$x,
                                kde[[g]]$y,
                                kde[[g]]$z,
                                level=c(quant[[g]][1]))
    for(i in 1:length(contours[[g]])){
      polygon(contours[[g]][[i]]$x,
              contours[[g]][[i]]$y,
              border=NA,
              col=fc[g]
```

```
        )
      }
    }
  }

   for(g in 1:length(groups)){
    if(nrow(groups[[g]])>1){
      points(groups[[g]][,xx],groups[[g]][,yy],
             pch=21,
             col=NA,
             cex=1,
             bg=fc[g])
    }

    if(nrow(groups[[g]])==1){
      text(groups[[g]][,xx],groups[[g]][,yy],
           levels(ts)[g],
           cex=0.5,
           col=fc[g]
      )
    }
    else {
      text(colMeans(groups[[g]])[xx],colMeans(groups[[g]])[yy],
           levels(ts)[g],
           cex=0.5,
           col="black"
      )
    }
  }
  arrows(0,0,loadings[,xx],loadings[,yy],
         length=0.1,
         angle=15
  )
  text(loadings[,xx]*1.2,loadings[,yy]*1.2,rownames(loadings),
       cex=0.7,
       col="darkblue",
       font=2
  )

  list(groups=groups, pc=pca)
}
```

pcaplot.clr kde()---unstandardardized PCA plot with KDEs

This function generates a PCA plot of unstandardized data with loadings and scores normalized to the eigenvalues. It is designed for handling clr-transformed data, though loading vectors may require alternative scaling. It also generates 100% KDE contours around defined training sets.

Function arguments:

- *x = data*
- *xx = PC axis to plot on the abscissa (1, 2, 3, ...)*
- *yy = PC axis to plot on the ordinate (1, 2, 3, ...)*
- *ts = a vector for defining specimen groups or tranings sets (members of the same group will be assigned the same colour)*
- *fc = a vector of field colour choices for points and 2D KDE contours for each training set*

Function Values:

- *groups = data clustered by grouping variable*
- *pc = results of the PCA by* ***prcomp()***

```
pcaplot.clr.kde<-function(x,xx,yy,ts,fc)
{
  pca<-prcomp(x,retx=T,scale=F)
  scores<-array(data=NA,
                dim=c(nrow(pca$x),ncol(pca$x)),
                dimnames=list(rownames(pca$x),colnames(pca$x))
                )
  for (l in 1:ncol(pca$x)){
    scores[,l]<-pca$x[,l]/pca$sdev[l]
  }
  loadings<-array(data=NA,
                  dim=c(nrow(pca$rotation),ncol(pca$rotation)),
                  dimnames=list(rownames(pca$rotation),colnames(pca$rotation)
)
                  )
  for (m in 1:ncol(pca$rotation)){
    loadings[,m]<-pca$rotation[,m]*pca$sdev[m]
  }
  percx<-round(pca$sdev[xx]^2/sum(pca$sdev^2)*100,2)
  percy<-round(pca$sdev[yy]^2/sum(pca$sdev^2)*100,2)

  groups<-list()
  for(i in 1:nlevels(ts)){
```

```
    groups[[i]]<-subset(scores,ts==levels(ts)[i])
  }
  widths<-array(data=NA,c(length(groups),2))
  for(j in 1:length(groups)){
    ifelse(nrow(groups[[j]])<4,
           widths[j,1]<-NA,
           widths[j,1]<-width.SJ(groups[[j]][,xx],method="dpi")
    )
    ifelse(nrow(groups[[j]])<4,
           widths[j,2]<-NA,
           widths[j,2]<-width.SJ(groups[[j]][,yy],method="dpi")
    )
  }
  kde<-list()
  for(k in 1:length(groups)){
    if(is.na(widths[k,1])){
      kde[[k]]<-NA
    }
    else {
      kde[[k]]<-kde2d(as.vector(groups[[k]][,xx]),as.vector(groups[[k]][,yy])
,
                      h=c(widths[k,1],widths[k,2]),
                      n=c(1000,1000),
                      lims=c(range(scores[,xx]-5,scores[,xx]+5),
                             range(scores[,yy]-5,scores[,yy]+5))
      )
    }
  }

  pkde<-list()
  for(p in 1:length(groups)){
    if(is.na(widths[p,1])){
      pkde[[p]]<-NA
    }
    else {
      pkde[[p]]<-pkde2d(groups[[p]][,xx],groups[[p]][,yy],
                        h=c(widths[p,1],widths[p,2]),
                        n=c(1000,1000),
                        lims=c(range(scores[,xx]-5,scores[,xx]+5),
                               range(scores[,yy]-5,scores[,yy]+5))
      )
    }
```

```
  }

  qe<-seq(0,1,len=10001)

  quant<-list()
  n_groups<-NULL
  for (p in 1:length(groups)){
    if(is.na(widths[p,1])){
      quant[[p]]<-NA
    }
    else {
      n_groups[p]<-nrow(groups[[p]])
      ptdens<-NULL
      for(k in 1:n_groups[p]){
        ptdens<-rbind(ptdens,pkde[[p]]$z[k,k])
      }
      quant[[p]]<-quantile(ptdens,probs=qe)
    }
  }

  plot(scores[,xx],scores[,yy],
       pch=NA,
       xlim=c(min(scores[,xx],loadings[,xx])-0.5,max(scores[,xx],loadings[,xx
])+0.5),
       ylim=c(min(scores[,yy],loadings[,yy])-0.5,max(scores[,yy],loadings[,yy
])+0.5),
       xlab=paste(colnames(scores)[xx],"(",percx,"%)",sep=""),
       ylab=paste(colnames(scores)[yy],"(",percy,"%)",sep="")
  )

  contours<-list()
  for(g in 1:length(groups)){
    if(is.na(widths[g,1])){
      next
    }
    else {
      contours[[g]]<-contourLines(kde[[g]]$x,
                                  kde[[g]]$y,
                                  kde[[g]]$z,
                                  level=c(quant[[g]][1]))
      for(i in 1:length(contours[[g]])){
        polygon(contours[[g]][[i]]$x,
```

```
              contours[[g]][[i]]$y,
              border=NA,
              col=fc[g]
      )
    }
  }
}

for(g in 1:length(groups)){
  if(nrow(groups[[g]])>1){
    points(groups[[g]][,xx],groups[[g]][,yy],
           pch=21,
           col=NA,
           cex=2,
           bg=fc[g])
  }

  if(nrow(groups[[g]])==1){
    text(groups[[g]][,xx],groups[[g]][,yy],
         levels(ts)[g],
         cex=0.8,
         col=fc[g]
    )
  }
  else {
    text(colMeans(groups[[g]])[xx],colMeans(groups[[g]])[yy],
         levels(ts)[g],
         cex=0.7,
         col="black"
    )
  }
}
arrows(0,0,loadings[,xx],loadings[,yy],
       length=0.1,
       angle=15
)
text(loadings[,xx]*1.2,loadings[,yy]*1.2,rownames(loadings),
     cex=0.8,
     col="darkblue",
     font=2
)
```

```
  list(groups=groups, pc=pca)
}
```

ternary() --- ternary diagram

This function is an adaptation of the **ternary()** function found in the StatDA package (Filzmoser, 2015). It plots the relative proportions of three component data into a ternary or three pole diagram.

Function arguments:

- *x = matrix or dataframe with three columns; each row must sum to 100*
- *group = a grouping variable*
- *nam = variable names*
- *grid = a logical argument specifying whether or not to draw a grid*
- *... = further graphical parameters*

Function values:

- *fc = a vector of field colours assigned to each group*

*See **ternary()** in package StatDA (Filzmoser, 2015).*

```
ternary<-function (x, group, nam = NULL, grid = FALSE, ...)
{
  val = 0.6
  if (is.null(nam)) {
    nam <- dimnames(x)[[2]]
  }
  s <- rowSums(x)
  if (any(s <= 0))
    stop("each row of the input `object' must have a positive sum")
  dat <- x/s
  xp <- dat[, 2] + dat[, 3]/2
  yp <- dat[, 3] * sqrt(3)/2
  par(pty = "s")

  clusters<-list()
  for(i in 1:nlevels(group)){
    clusters[[i]]<-dat[group==levels(group)[i],]
  }
```

```
  fc<-rainbow(length(clusters),s=1,v=1,alpha=0.3)

  plot(xp, yp, xlim = c(0, 1), ylim = c(0, 0.9),  pch=NA, frame.plot = FALSE,

       xaxt = "n", yaxt = "n", xlab = "", ylab = "", ...)
  segments(0, 0, 1, 0)
  segments(0, 0, 1/2, sqrt(3)/2)
  segments(1/2, sqrt(3)/2, 1, 0)
  mtext(nam[1], side = 1, line = -1, at = -0.05, cex = 0.6)
  mtext(nam[2], side = 1, line = -1, at = 1.05, cex = 0.6)
  text(0.5, 0.9, nam[3], cex = 0.6)

  for(g in 1:length(clusters)){
    if(is.null(nrow(clusters[[g]]))==F){
      points(clusters[[g]][,2]+clusters[[g]][,3]/2,clusters[[g]][,3]*sqrt(3)/
2,
             pch=21,
             cex=2,
             col="black",
             bg=fc[g])}
    else{
      points(clusters[[g]][,2]+clusters[[g]][,3]/2,clusters[[g]][,3]*sqrt(3)/
2,
             pch=21,
             cex=2,
             col=NA,
             bg=fc[g])
    }
  }

  if (grid == TRUE) {
    segments(0.2, 0, 0.1, sqrt(0.03), col = grey(val), lty = "dashed")
    segments(0.4, 0, 0.2, sqrt(0.12), col = grey(val), lty = "dashed")
    segments(0.6, 0, 0.3, sqrt(0.27), col = grey(val), lty = "dashed")
    segments(0.8, 0, 0.4, sqrt(0.48), col = grey(val), lty = "dashed")
    segments(0.2, 0, 0.6, sqrt(0.48), col = grey(val), lty = "dashed")
    segments(0.4, 0, 0.7, sqrt(0.27), col = grey(val), lty = "dashed")
    segments(0.6, 0, 0.8, sqrt(0.12), col = grey(val), lty = "dashed")
    segments(0.8, 0, 0.9, sqrt(0.03), col = grey(val), lty = "dashed")
    segments(0.1, sqrt(0.03), 0.9, sqrt(0.03), col = grey(val),
             lty = "dashed")
```

```
    segments(0.2, sqrt(0.12), 0.8, sqrt(0.12), col = grey(val),
             lty = "dashed")
    segments(0.3, sqrt(0.27), 0.7, sqrt(0.27), col = grey(val),
             lty = "dashed")
    segments(0.4, sqrt(0.48), 0.6, sqrt(0.48), col = grey(val),
             lty = "dashed")

    text(0.95, 0.21, "20", col = grey(val), cex = 0.6, srt = 60)
    text(0.86, 0.35, "40", col = grey(val), cex = 0.6, srt = 60)
    text(0.75, 0.54, "60", col = grey(val), cex = 0.6, srt = 60)
    text(0.64, 0.72, "80", col = grey(val), cex = 0.6, srt = 60)
    text(0.05, 0.21, "80", col = grey(val), cex = 0.6, srt = 300)
    text(0.14, 0.35, "60", col = grey(val), cex = 0.6, srt = 300)
    text(0.25, 0.54, "40", col = grey(val), cex = 0.6, srt = 300)
    text(0.36, 0.72, "20", col = grey(val), cex = 0.6, srt = 300)
    text(0.2, -0.02, "20", col = grey(val), cex = 0.6, srt = 60)
    text(0.4, -0.02, "40", col = grey(val), cex = 0.6, srt = 60)
    text(0.6, -0.02, "60", col = grey(val), cex = 0.6, srt = 60)
    text(0.8, -0.02, "80", col = grey(val), cex = 0.6, srt = 60)

  }
  list(fc=fc)
}
```

Acknowledgements

The graphing and related functions above, especially the use of KDE's in the display of groups, were adapted and developed thanks to support by a Marie Curie International Incoming Fellowship (PIIF-GA-2013-624448) within the 7th European Community Framework Programme as part of the IRONWORKS project. Special thanks go to Tom Birch and Marcos Martinón-Torres for discussion surrounding the use of KDEs. I am grateful to Jane Humphris for her interest in applying these graphing approaches to materials in the Sudan Project at UCL Qatar.

References

Filzmoser, P., 2015. StatDA: Statistical Analysis for Environmental Data.

Venables, W.N., Ripley, B.D., 2002. Modern Applied Statistics with S, Fourth. ed. Springer, New York.